Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Electrical Engineering Department

Microprocessors (22344)
Software Quiz App Assembly Project

Author: Qais Jildeh (20210155)

Supervisor: Dr. Heba Abdel-Nabi

August 6, 2024

Abstract

This project deploys a variable Multiple-Choice Quiz Application in 8086 assembly language with user login, randomly sorted questions, and scoring. Users are verified through preassigned credentials, respond dynamically requested questions on the selected subject matter, and get instant feedback. The solution focuses on modular design, memory management, and interrupt-based I/O, demonstrating deployed low-level programming to interactive software.

Table of Contents

Table of Figures

Introduction

This project creates an 8086-assembly language customizable quiz game about the cars movie with user authentication, dynamic question shuffling, and scoring. Based on modular procedures as its basic structure, it exhibits effective memory management and input handling in real-time, demonstrating real-world uses of low-level programming for interactive systems.

Requirements

- User Authentication: Develop a login system with predefined credentials and a 3-attempt limit.
- Dynamic Question Handling: Load, shuffle, and display 4 MCQs on a user-chosen topic.
- Scoring System: Calculate and display user scores.
- Modular Design: Organize code into reusable procedures and analyze performance efficiency.

Program

The quiz app consists of the procedures **userAuthentication**, **shuffleQuiz**, **quiz**, **choose**, **isValidAnswer**, **gradeQuiz**, **printScoreboard**, and **terminateProgram** to provide it with the correct logic to function according to the requirements. The program will have the following the flow of execution.

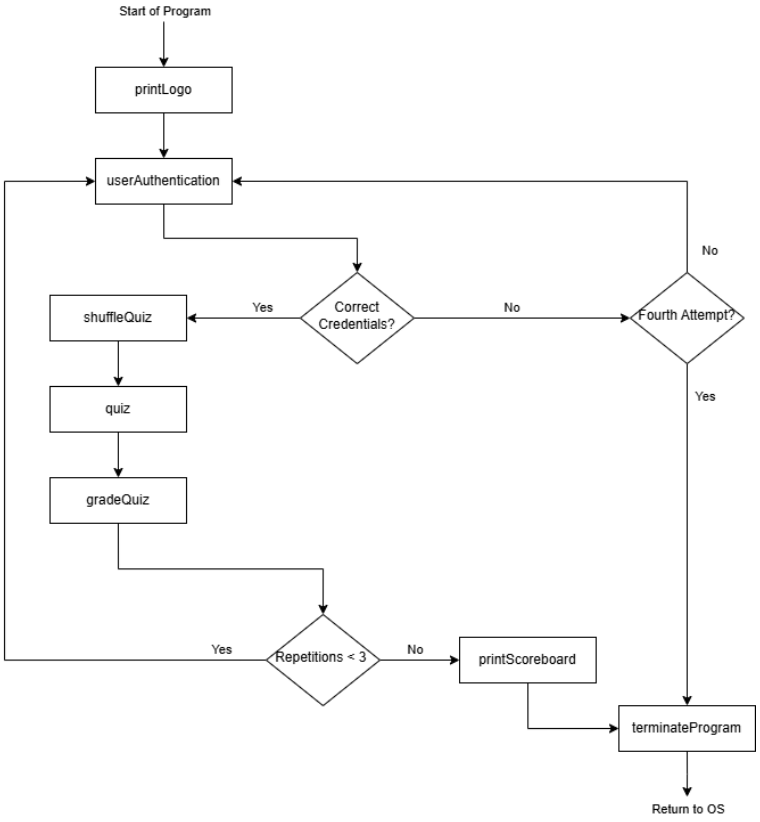

Figure 1: Quiz app flowchart

The program begins by printing the quiz logo, then prompts the user to enter their credentials. If the credentials are entered correctly, the quiz starts; otherwise, the user has two more attempts to log in. After successful authentication, the questions are shuffled using the **shuffleQuiz** procedure, and the user receives one question at a time. For each question, the user is prompted to input a choice. If the input is invalid, they will be prompted

again until a valid choice is entered. Once a valid choice is provided, it is recorded and the score is updated accordingly. After the user answers 4 questions (selected from a pool of 5), the final score is displayed, and the authentication process begins again for the next user. This process repeats for a total of 3 users. Afterward, the scoreboard is updated and the program terminates.

```
call printLogo
mov cx, 3
restart:
cld
push cx
call userAuthentication
call shuffleQuiz
call quiz
call gradeQuiz
pop cx
loop restart
call printScoreboard
call terminateProgram
```

Figure 2: Quiz app execution flow

Procedure #1: println

println procedure handles printing of sequence of newline as output formatting. It uses DOS interrupt 21h subroutine 09h to print the string in memory located at address of endl containing ASCII strings 13 and 10 for carriage return and line feed, respectively, and which place cursor back on line beginning and move it one line down. This process is significant in maintaining output readable throughout the program because it keeps getting invoked following the outputting of messages, queries, or user prompts in order to prevent text overlap on the screen.

```
endl db 13, 10, '$'  ; 13 ==> carrige return (cursor at start), 10 ==> new line
(move cursor downwards in same column)
```

Figure 3: Variables used by println procedure in the data segment

```
; --- [Print New Line] ---
    println proc
        lea dx, endl
        mov ah, 0x09
        int 21h

        ret
    println endp
```

Figure 4: println Procedure

Procedure #2: printLogo

This routine draws an ASCII art styled logo when the program starts. It prints eight predefined strings (logoLine1 to logoLine8) sequentially with the DOS string output call (09h). Each line is topped off with a call to println to maintain vertical spacing. The logo is an opening graphical introduction of the program and is succeeded by a separator line (logoLine8) to introduce the authentication phase. The official logo emphasizes the focus of the program on user experience.

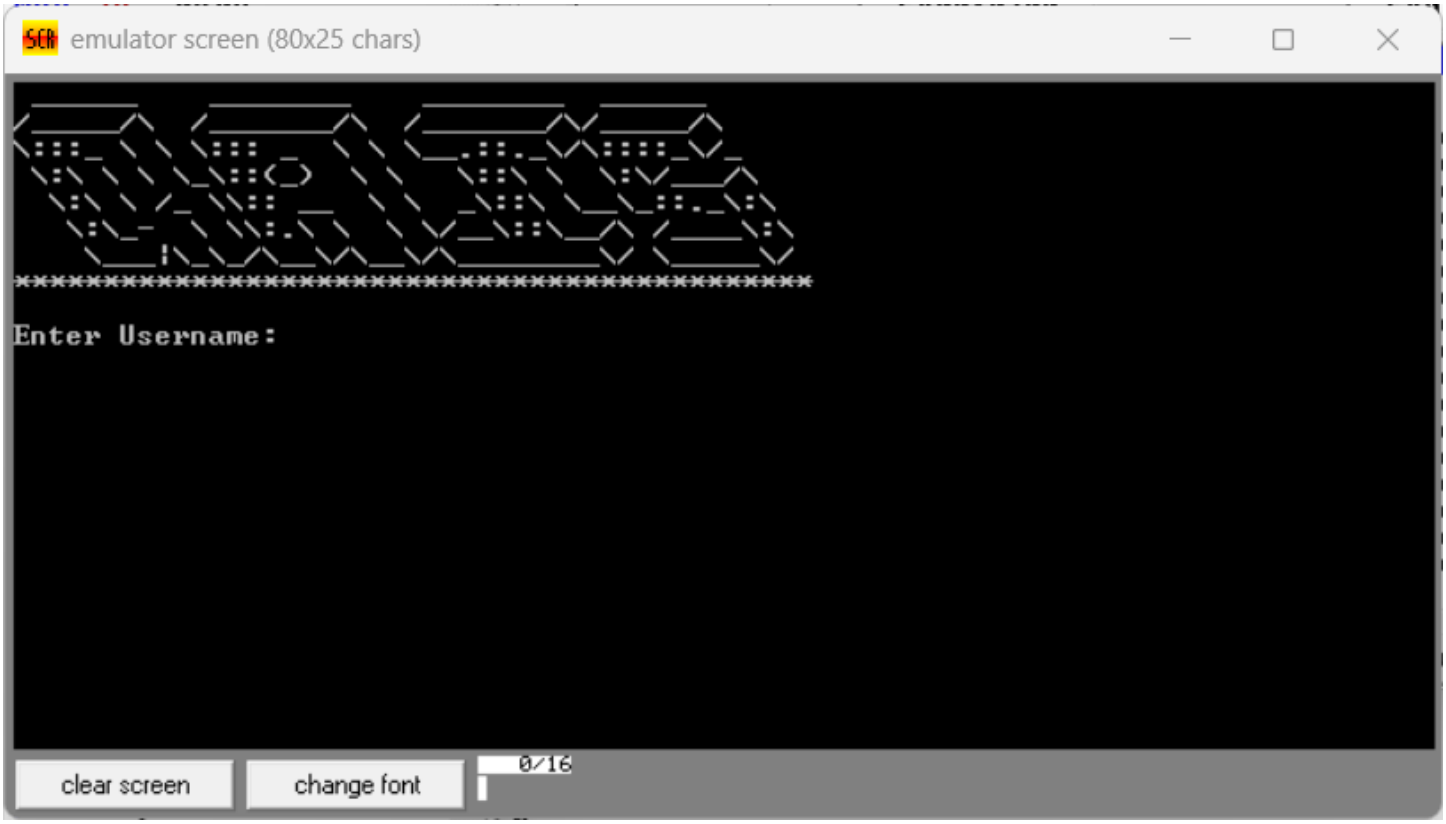

Figure 5: printLogo procedure output

```
; logo
    logoLine1 db ' ______    ________    ________  ______       $'
    logoLine2 db '/_____/\  /_______/\  /_______/\/_____/\     $'
    logoLine3 db '\:::_ \ \ \::: _  \ \ \__.::._\/\::::_\/_    $'
    logoLine4 db ' \:\ \ \ \_\::(_)  \ \   \::\ \  \:\/___/\   $'
    logoLine5 db '  \:\ \ /_ \\:: __  \ \  _\::\ \__\_::._\:\  $'
    logoLine6 db '   \:\_-  \ \\:.\ \  \ \/__\::\__/\ /____\:\ $'
    logoLine7 db '    \___|\_\_/\__\/\__\/\________\/ \_____\/ $'
    logoLine8 db '*****************************************$'
```

Figure 6: Variables used by printLogo procedure in the data segment

```
; --- [Print Starting Screen Logo] ---
    printLogo proc
        lea dx, logoLine1
        mov ah, 0x09
        int 21h
        call println
        lea dx, logoLine2
        mov ah, 0x09
        int 21h
        call println
        lea dx, logoLine3
        mov ah, 0x09
        int 21h
        call println
        lea dx, logoLine4
        mov ah, 0x09
        int 21h
        call println
        lea dx, logoLine5
        mov ah, 0x09
        int 21h
        call println
        lea dx, logoLine6
        mov ah, 0x09
        int 21h
        call println
        lea dx, logoLine7
        mov ah, 0x09
        int 21h
        call println
        lea dx, logoLine8
        mov ah, 0x09
        int 21h
        call println
        ret
    printLogo endp
```

Figure 7: printLogo procedure

Procedure #3: userAuthentication

The authentication takes place by ensuring user credentials during three attempts. It requests user input of the username and password from 09h to give output prompts (userRequest, passRequest) and 0Ah to return buffered input as userIn and passIn. The operation also compares input strings with hardcoded user credentials (user1, user2, user3 and corresponding passwords) through the cmpsb instruction, performing a byte-by-byte comparison. The input size (stored in the second byte of the buffer) scales the comparison range. On successful validation, the process outputs a success message; after three failures, it terminates the program.

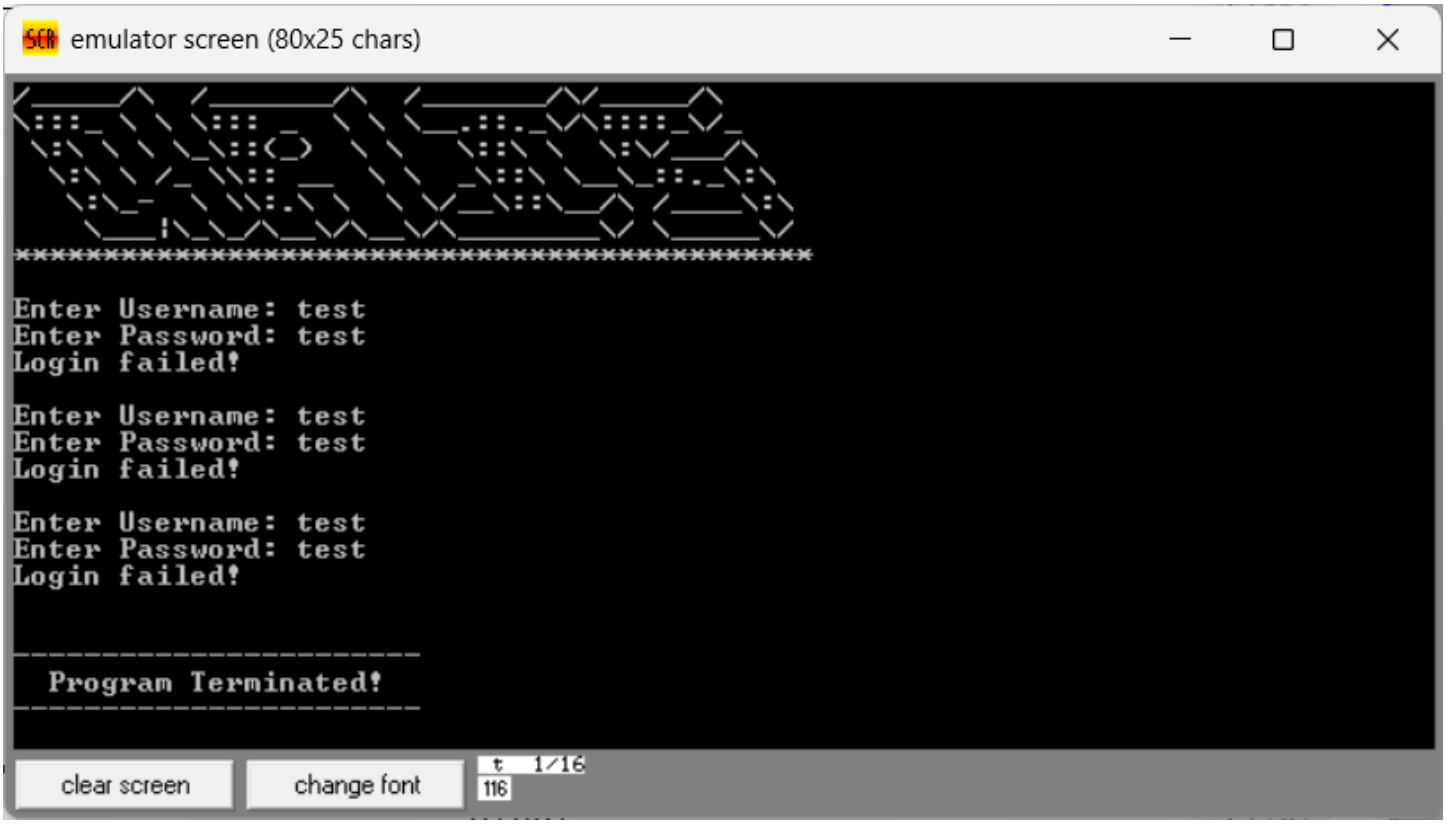

Figure 8: Output after three wrong login attempts

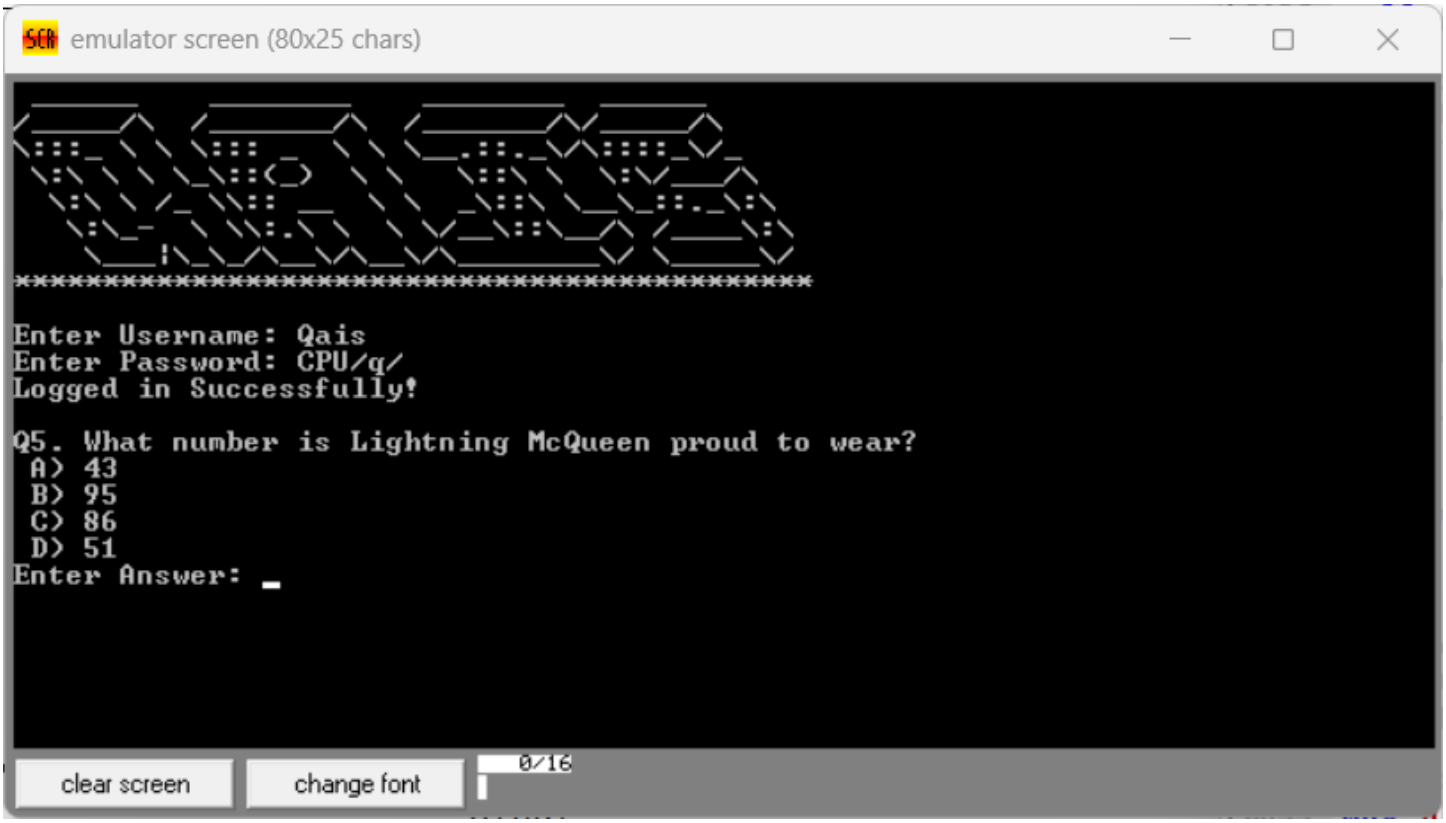

Figure 9:Output after entering correct credentials

```
; login information
    userRequest db 'Enter Username: $'
    passRequest db 'Enter Password: $'
    user1 db 'Qais$'
    pass1 db 'CPU/q/$'
    user2 db 'Bassem$'
    pass2 db 'ALU/b/$'
    user3 db 'Hanna$'
    pass3 db 'GPU/h/$'

    userIn db 7
           db 0
           db 7 dup('$')

    passIn db 7
           db 0
           db 7 dup('$')

    success db 'Logged in Successfully!$'
    fail db 'Login failed!$'
```

Figure 10: Variables used by userAuthentication procedure in the data segment

```
; --- [User Authentication] ---
    userAuthentication proc
        call println

        cld
        mov cx, 3
        authentication:
        push cx
        ; print enter username statement
        lea dx, userRequest
        mov ah, 0x09
        int 21h

        ; take 6 characters
        lea dx, userIn
        mov ah, 0x0A
        int 21h

        call println

        ; print enter password statement
        lea dx, passRequest
        mov ah, 0x09
        int 21h

        ; enter 6 character password
        lea dx, passIn
        mov ah, 0x0A
        int 21h

        call println


        testUsername1:
        lea si, user1
        lea di, userIn
        mov cl, [di + 1]
        xor ch, ch
        add di, 2
        rep cmpsb
        je testPassword1
        jmp testUsername2

        testPassword1:
        lea si, pass1
```

```
lea di, passIn
mov cl, [di + 1]
xor ch, ch
add di, 2
rep cmpsb
je successful
jmp failed


testUsername2:
lea si, user2
lea di, userIn
mov cl, [di + 1]
xor ch, ch
add di, 2
rep cmpsb
je testPassword2
jmp testUsername3

testPassword2:
lea si, pass2
lea di, passIn
mov cl, [di + 1]
xor ch, ch
add di, 2
rep cmpsb
je successful
jmp failed


testUsername3:
lea si, user3
lea di, userIn
mov cl, [di + 1]
xor ch, ch
add di, 2
rep cmpsb
je testPassword3
jmp failed

testPassword3:
lea si, pass3
lea di, passIn
mov cl, [di + 1]
xor ch, ch
```

```
        add di, 2
        rep cmpsb
        je successful
        jmp failed


        failed:
        lea dx, fail
        mov ah, 0x09
        int 21h

        call println

        call println

        pop cx
        loop authentication

        call terminateProgram

        successful:
        lea dx, success
        mov ah, 0x09
        int 21h

        call println

        pop cx

        ret
    userAuthentication endp
```

Figure 11: userAuthentication procedure

To calculate the number of clock cycles, we need to analyze the code first. In the worst case, the authentication procedure will be executed three consecutive times due to login error. Here is the analysis of the procedure:

Everytime we call **println** it will cost us the time of calling, executing the procedure, and returning. Number of clock cycles is as follows

$$Clock\ Cycles\ for\ println = 19 + 2 + 6 + 4 + 51 + 8 = 90$$

Before entering the loop, we do **mov, cx, 3** this cost us 4 clock cycles.

The number of clock cycles inside the loop, note the with each time we take user input or output to the display it will cost us (2+6+4+51 = 63 clock cycles):

$$Clock\ Cycles\ inside\ the\ authentication\ loop$$
$$= 11 + 5(63) + 4(90) + 3(2 + 6 + 2 + 6 + 8 + 12 + 3 + 4 + 7 * 3 + 15) + 8 + 17 = 948$$

The loop will be entered 3 times but the loop instruction will take branch twice only:

$$Total\ clock\ cycles\ by\ the\ authentication\ loop = 3(948) - 17 + 5 = 2832$$

The procedure will call the **terminateProgram** procedure, this will cost us:

$$Clock\ Cycles = 4(90) + 3(63) + 4 + 51 = 604$$

Total number of clock cycles will be 604 + 2832 = 3436 clock cycles.

The 8086 has a 5MHz clock frequency, so the time need to execute the procedure is calculated as follows:

$$Time\ needed\ to\ execute\ authentication = 200ns \times 3436 = 687.2\mu s$$

Procedure #4: shuffleQuiz

This process pseudo-randomizes question order of quizzes with a pseudo-random number generator. It initializes by producing a random index (0–4) based on a seed. The seed is multiplied by 4, divided by 5, and its remainder (which is stored in AH) is used to create the index. According to the index, pair-wise elements within the items array (holding question numbers 1–5) are swapped (Swap00 interchanges the second and third). The seed is then converted with the formula seed = (seed * 7) + 3 to introduce randomness for future shuffles.

```
; Shuffling
    items db 1, 2, 3, 4, 5
    seed db 113
```

Figure 12: Variables used by shuffleQuiz procedure in the data segment

```
; --- [Shuffling Algorithm] ---
    shuffleQuiz proc
        ShuffleLoop:
        ; Generate random index (0-4)
        MOV AL, Seed             ; Current seed value
        MOV AH, 0                ; Clear AH for multiplication
        MOV CL, 4                ; Scaling factor
        MUL CL                   ; AX = AL * 4
        MOV BL, 5                ; Modulo 5
        DIV BL                   ; AX / 5 ? AL=quotient, AH=remainder
        MOV DL, AH               ; Random index (0-4) in DL

        ; Swap Items[DL] with another item based on DL
        MOV SI, OFFSET Items     ; Address of Items array

        CMP DL, 0
        JE Swap00
        CMP DL, 1
        JE Swap01
        CMP DL, 2
        JE Swap02
        CMP DL, 3
        JE Swap03
        CMP DL, 4
        JE Swap04

        Swap00:                  ; Swap Items[0] and Items[1]
        MOV AL, [SI]
        MOV BL, [SI + 1]
        MOV [SI], BL
        MOV [SI + 1], AL
        JMP DoneSwap

        Swap01:                  ; Swap Items[1] and Items[2]
        MOV AL, [SI + 1]
        MOV BL, [SI + 2]
        MOV [SI + 1], BL
        MOV [SI + 2], AL
        JMP DoneSwap

        Swap02:                  ; Swap Items[2] and Items[3]
        MOV AL, [SI + 2]
        MOV BL, [SI + 3]
        MOV [SI + 2], BL
        MOV [SI + 3], AL
```

```
    JMP DoneSwap

    Swap03:                 ; Swap Items[3] and Items[4]
    MOV AL, [SI + 3]
    MOV BL, [SI + 4]
    MOV [SI + 3], BL
    MOV [SI + 4], AL
    JMP DoneSwap

    Swap04:                 ; Swap Items[0] and Items[4]
    MOV AL, [SI]
    MOV BL, [SI + 4]
    MOV [SI], BL
    MOV [SI + 4], AL
    JMP DoneSwap

    DoneSwap:
    ; Update seed for next iteration
    MOV AL, Seed
    MOV BL, 7
    MUL BL                  ; AL = AL * 7
    ADD AL, 3               ; AL = AL + 3
    MOV Seed, AL            ; Store updated seed

    ret
shuffleQuiz endp
```

Figure 13: shuffleQuiz Procedure

Procedure #5: quiz

The quiz process provides the shuffled questions. It cycles through the items array and proceeds to the related question label (question1) based on the shuffled index. Each question label displays the question text and multiple-choice items (q1–q5 and a1–d5) with 09h. The currentQuestion variable stores the current question to enable the choose process to retrieve the user's answer to the right question.

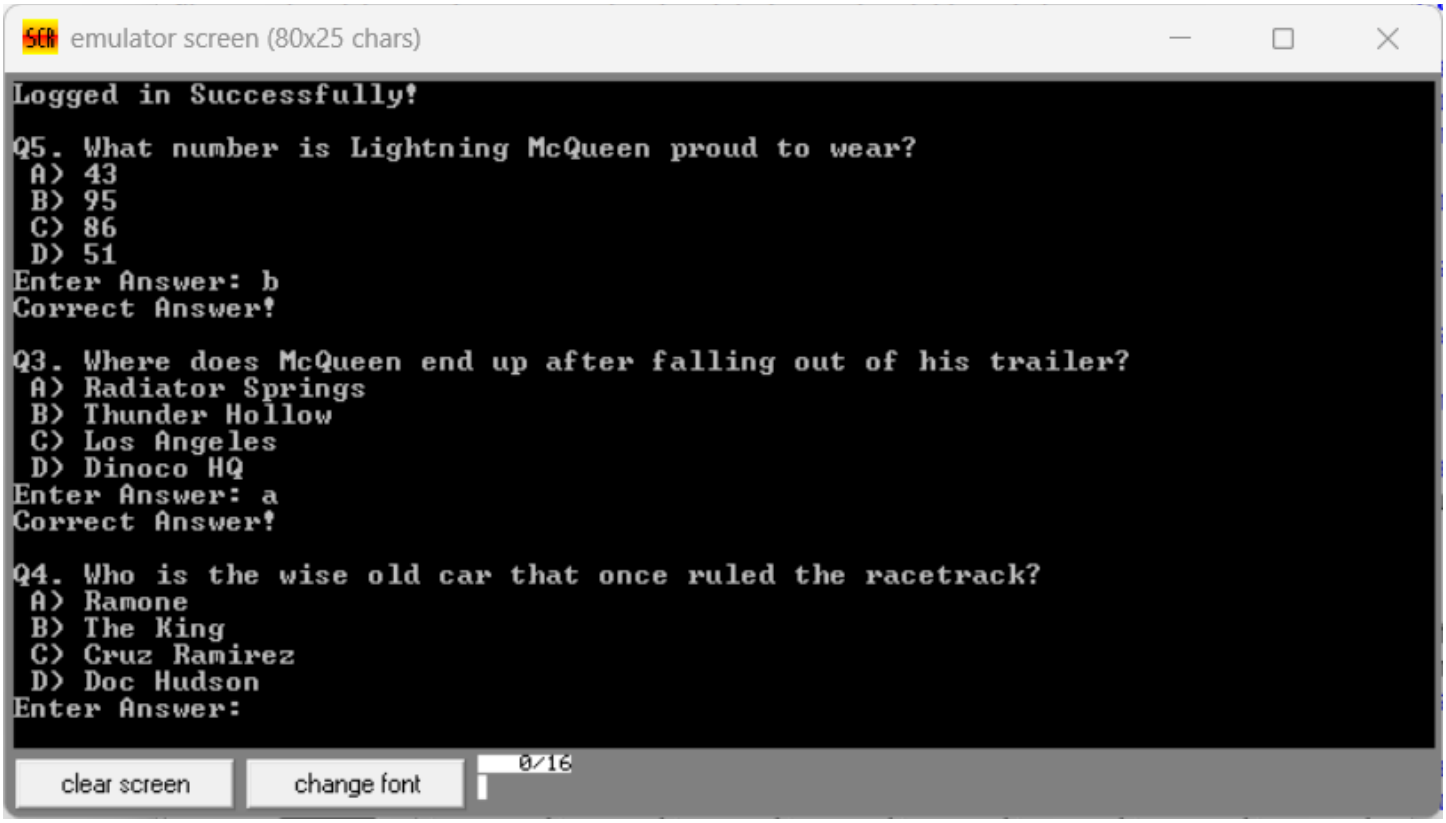

Figure 14: quiz procedure output

```
; Questions
    q1 db 'Q1. Who zooms through the story as the star of Cars?$'
    a1 db ' A) Lightning McQueen$'
    b1 db ' B) Tow Mater$'
    c1 db ' C) Doc Hudson$'
    d1 db ' D) Chick Hicks$'

    q2 db 'Q2. What kind of vehicle is Mater, Lightning's goofy best bud?$'
    a2 db ' A) A race car$'
    b2 db ' B) A monster truck$'
    c2 db ' C) A tow truck$'
    d2 db ' D) A fire truck$'

    q3 db 'Q3. Where does McQueen end up after falling out of his trailer?$'
    a3 db ' A) Radiator Springs$'
    b3 db ' B) Thunder Hollow$'
    c3 db ' C) Los Angeles$'
    d3 db ' D) Dinoco HQ$'

    q4 db 'Q4. Who is the wise old car that once ruled the racetrack?$'
    a4 db ' A) Ramone$'
    b4 db ' B) The King$'
    c4 db ' C) Cruz Ramirez$'
    d4 db ' D) Doc Hudson$'

    q5 db 'Q5. What number is Lightning McQueen proud to wear?$'
    a5 db ' A) 43$'
    b5 db ' B) 95$'
    c5 db ' C) 86$'
    d5 db ' D) 51$'

    correctAnswer db 'Correct Answer!$'
    wrongAnswer db 'Wrong Answer!$'
    answerPrompt db 'Enter Answer: $'

    ans1 db 2 dup('$')
    ans2 db 2 dup('$')
    ans3 db 2 dup('$')
    ans4 db 2 dup('$')
    ans5 db 2 dup('$')

    currentQuestion db 0
```

Figure 15: Variables used by quiz procedure in the data segment

```
; --- [The Quiz] ---
    quiz proc
        lea si, [items]
        mov cx, 0x0004
        cld
        nextQuestion:
        mov bx, si
        add bx, cx
        mov bl, [bx]
        xor bh, bh

        push cx
        cmp bl, 1
        je question1

        cmp bl, 2
        je question2

        cmp bl, 3
        je question3

        cmp bl, 4
        je question4

        cmp bl, 5
        je question5

        backToLoop:
        pop cx
        loop nextQuestion

        ret

        question1:
        call println

        lea dx, q1
        mov ah, 0x09
        int 21h

        call println

        lea dx, a1
        mov ah, 0x09
```

```
int 21h

call println

lea dx, b1
mov ah, 0x09
int 21h

call println

lea dx, c1
mov ah, 0x09
int 21h

call println

lea dx, d1
mov ah, 0x09
int 21h

mov [currentQuestion], 0x01   ; save question number

call choose

call println

jmp backToLoop


question2:
call println

lea dx, q2
mov ah, 0x09
int 21h

call println

lea dx, a2
mov ah, 0x09
int 21h

call println

lea dx, b2
```

```
mov ah, 0x09
int 21h

call println

lea dx, c2
mov ah, 0x09
int 21h

call println

lea dx, d2
mov ah, 0x09
int 21h

mov [currentQuestion], 0x02   ; save question number

call choose

call println

jmp backToLoop


question3:
call println

lea dx, q3
mov ah, 0x09
int 21h

call println

lea dx, a3
mov ah, 0x09
int 21h

call println

lea dx, b3
mov ah, 0x09
int 21h

call println
```

```
lea dx, c3
mov ah, 0x09
int 21h

call println

lea dx, d3
mov ah, 0x09
int 21h

mov [currentQuestion], 0x03   ; save question number

call choose

call println

jmp backToLoop


question4:
call println

lea dx, q4
mov ah, 0x09
int 21h

call println

lea dx, a4
mov ah, 0x09
int 21h

call println

lea dx, b4
mov ah, 0x09
int 21h

call println

lea dx, c4
mov ah, 0x09
int 21h

call println
```

```
lea dx, d4
mov ah, 0x09
int 21h

mov [currentQuestion], 0x04   ; save question number

call choose

call println

jmp backToLoop


question5:
call println

lea dx, q5
mov ah, 0x09
int 21h

call println

lea dx, a5
mov ah, 0x09
int 21h

call println

lea dx, b5
mov ah, 0x09
int 21h

call println

lea dx, c5
mov ah, 0x09
int 21h

call println

lea dx, d5
mov ah, 0x09
int 21h
```

```
    mov [currentQuestion], 0x05   ; save question number

    call choose

    call println

    jmp backToLoop

    ret
quiz endp
```

Figure 16: quiz Procedure

Procedure #6: choose

This process stores and verifies the user's response. It outputs the answerPrompt string and inputs a single character through 01h, making the character visible on screen. Incorrect inputs or non-A/B/C/D characters produce two sound beeps (ASCII 07h) through 02h, and the loop will continue until the right answer is input. Correct answers are passed to isValidAnswer to be graded and update the score.

The **choose** procedure uses the same variables as the **quiz** segment.

```
; --- [Take Answer] ---
    choose proc
        incorrectInput:
        call println

        lea dx, answerPrompt
        mov ah, 0x09
        int 21h

        mov ah, 0x01
        int 21h

        call isValidAnswer
        cmp cx, 2
        je beepbeep
        jmp endOfLoop

        beepbeep:
        mov ah, 02h
        mov dl, 07h
        int 21h       ; first beep

        mov ah, 02h
        mov dl, 07h
        int 21h       ; second beep

        endOfLoop:
        loop incorrectInput

        ret
    choose end
```

Figure 17: choose Procedure

Procedure #7: isValidAnswer

The **isValidAnswer** procedure verifies answers and increments the score. It lowercase-transforms letters A–D via bitwise OR operation (or al, 0x20) and verifies if input is valid options. The answer is stored in buffers (ans1–ans5) depending on currentQuestion. Correct answers (e.g., a for Q1, c for Q2) increase the score variable, while incorrect answers activate the wrongAnswer message and beep. The score is represented as an ASCII digit (e.g., 48 for '0'), and this makes display logic simpler.

The **isValidAnswer** procedure also uses the same variables as **choose** and **quiz** procedures.

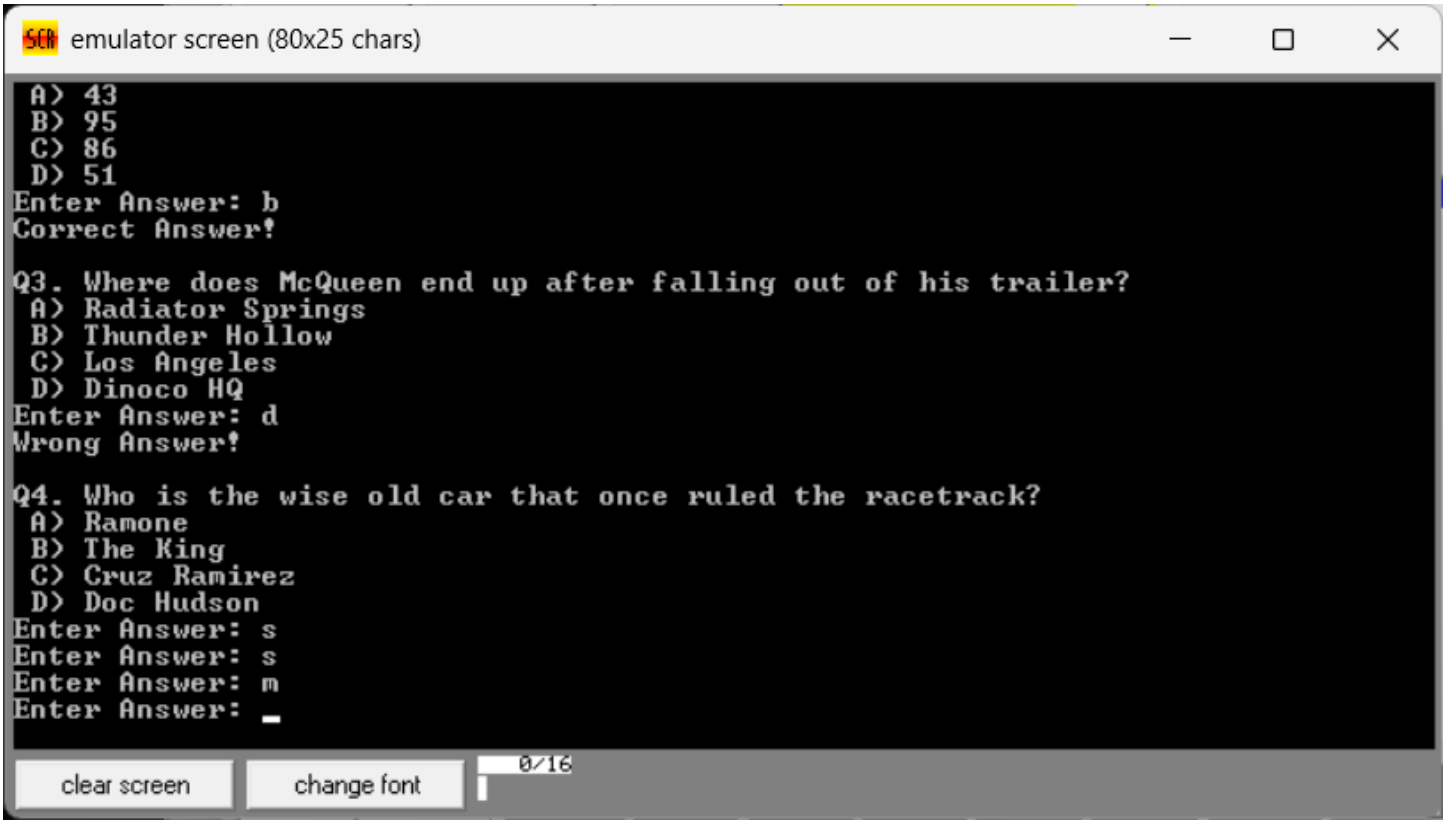

Figure 18: choose & isValidAnswer procedures output

```
; --- [Is Valid Answer] ---
    isValidAnswer proc
        cmp al, 'A'
        je toLower

        cmp al, 'a'
        je save

        cmp al, 'B'
        je toLower

        cmp al, 'b'
        je save

        cmp al, 'C'
        je toLower

        cmp al, 'c'
        je save

        cmp al, 'D'
        je toLower

        cmp al, 'd'
        je save

        mov cx, 0x0002
        ret

        toLower:
        or al, 0x20

        save:
        mov cx, 0x0001

        mov bl, currentQuestion

        cmp bl, 1
        je saveAnswerQ1

        cmp bl, 2
        je saveAnswerQ2

        cmp bl, 3
        je saveAnswerQ3
```

```
cmp bl, 4
je saveAnswerQ4

cmp bl, 5
je saveAnswerQ5

saveAnswerQ1:
mov [ans1], al
jmp updateScore

saveAnswerQ2:
mov [ans2], al
jmp updateScore

saveAnswerQ3:
mov [ans3], al
jmp updateScore

saveAnswerQ4:
mov [ans4], al
jmp updateScore

saveAnswerQ5:
mov [ans5], al
jmp updateScore

updateScore:
xor ah, ah
push ax

call println

pop ax

cmp bl, 1
jne questionScoring2
cmp al, 'a'
je addScore
jne incorrectChoiceChosen

questionScoring2:
cmp bl, 2
jne questionScoring3
cmp al, 'c'
```

```
je addScore
jne incorrectChoiceChosen

questionScoring3:
cmp bl, 3
jne questionScoring4
cmp al, 'a'
je addScore
jne incorrectChoiceChosen

questionScoring4:
cmp bl, 4
jne questionScoring5
cmp al, 'd'
je addScore
jne incorrectChoiceChosen

questionScoring5:
cmp bl, 5
jne incorrectChoiceChosen
cmp al, 'b'
je addScore
jne incorrectChoiceChosen

addScore:
mov dl, [score]
add dl, 1
mov [score], dl

lea dx, correctAnswer
mov ah, 0x09
int 21h

ret

incorrectChoiceChosen:
lea dx, wrongAnswer
mov ah, 0x09
int 21h

mov ah, 02h
mov dl, 07h
int 21h       ; first beep

mov ah, 02h
```

```
    mov dl, 07h
    int 21h       ; second beep

    ret
isValidAnswer endp
```

Figure 19: isValidAnswer Procedure

Procedure #8: gradeQuiz

Once the quiz is finished, the **gradeQuiz** procedure shows the old score and resets the score board. It prints statement1, the ASCII score (52 for '4'), and statement2 (/4!). The logged-in user is determined through string comparison, and his or her score is held in qaisScore, bassemScore, or hannaScore. The score variable is initialized to 48 ('0') in preparation for future quiz attempts. This modular solution enables multiple users to attempt the quiz one by one.

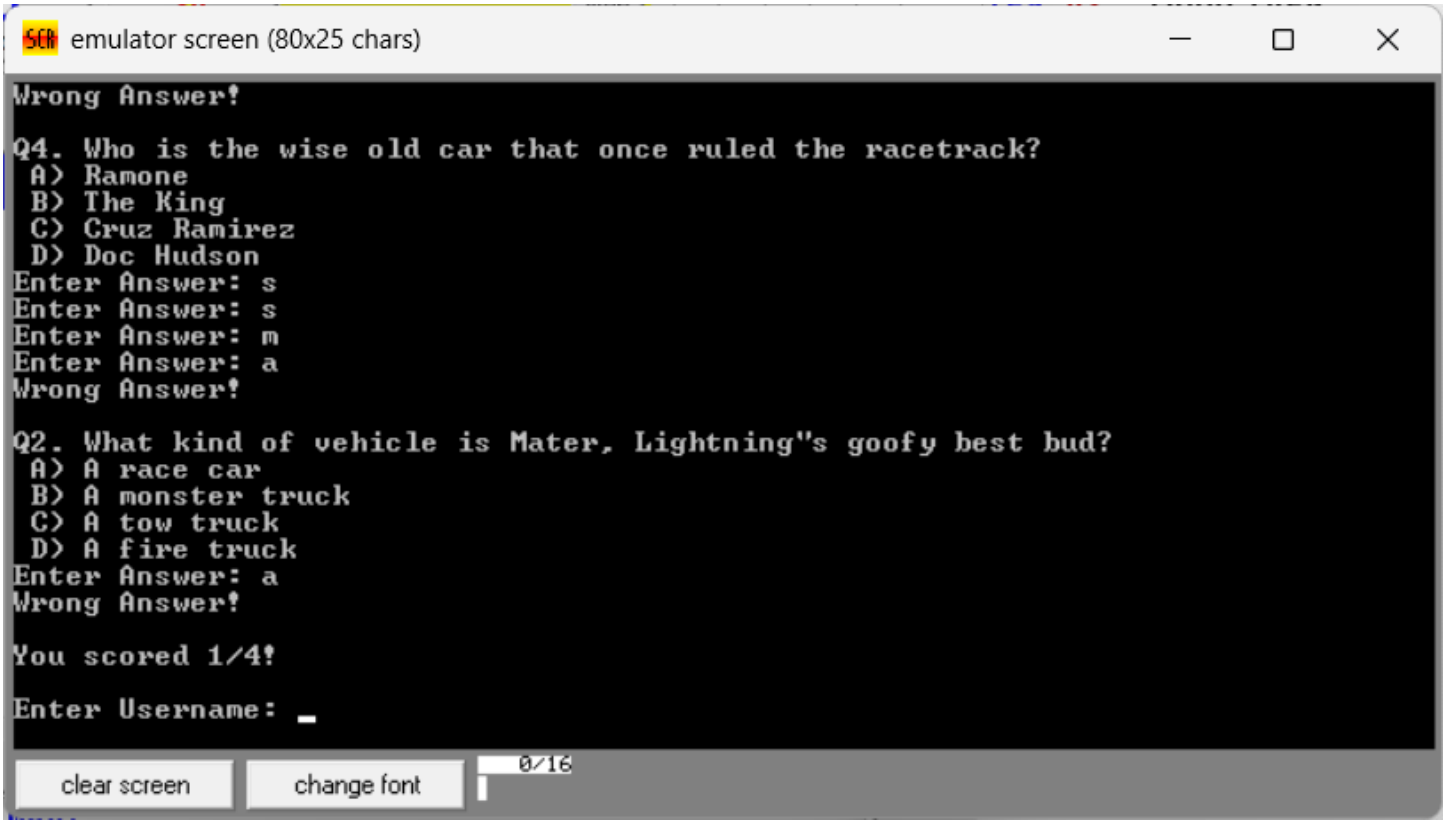

Figure 20: gradeQuiz procedure output

```
; Grading
    score db 48

    statement1 db 'You scored $'
    statement2 db '/4!$'
```

Figure 21: Variables used by gradeQuiz procedure in data segment

```
; --- [Grade Quiz] ---
    gradeQuiz proc
        call println

        lea dx, statement1
        mov ah, 0x09
        int 21h

        mov ah, 0x02
        mov dl, [score]
        int 21h

        lea dx, statement2
        mov ah, 0x09
        int 21h

        call println

        lea si, user1
        lea di, userIn
        mov cl, [di + 1]
        xor ch, ch
        add di, 2
        rep cmpsb
        je updateQaisScore
        jmp checkIfBassem

        checkIfBassem:
        lea si, user2
        lea di, userIn
        mov cl, [di + 1]
        xor ch, ch
        add di, 2
        rep cmpsb
        je updateBassemScore
        jmp checkIfHanna

        checkIfHanna:
        lea si, user3
        lea di, userIn
        mov cl, [di + 1]
        xor ch, ch
        add di, 2
        rep cmpsb
        je updateHannaScore
```

```
    ret

    updateQaisScore:
    mov bl, [score]
    mov [qaisScore], bl
    jmp finishedGrading

    updateBassemScore:
    mov bl, [score]
    mov [bassemScore], bl
    jmp finishedGrading

    updateHannaScore:
    mov bl, [score]
    mov [hannaScore], bl

    finishedGrading:
    mov bl, 48
    mov [score], bl

    ret
gradeQuiz endp
```

Figure 22: gradeQuiz procedure

Procedure #9: printScoreboard

The **printScoreboard** procedure ranks and prints scores of users in descending order. ASCII score values are compared to conditional jumps (jae, jb) for ranking the users. Pointers (firstName, secondName, thirdName) are allocated to user names in relation to their scores, and the outputs are printed out using colons and spaces to enhance formatting. The scoreboard increases competition by displaying rankings, utilizing 09h for strings and 02h for a single character.

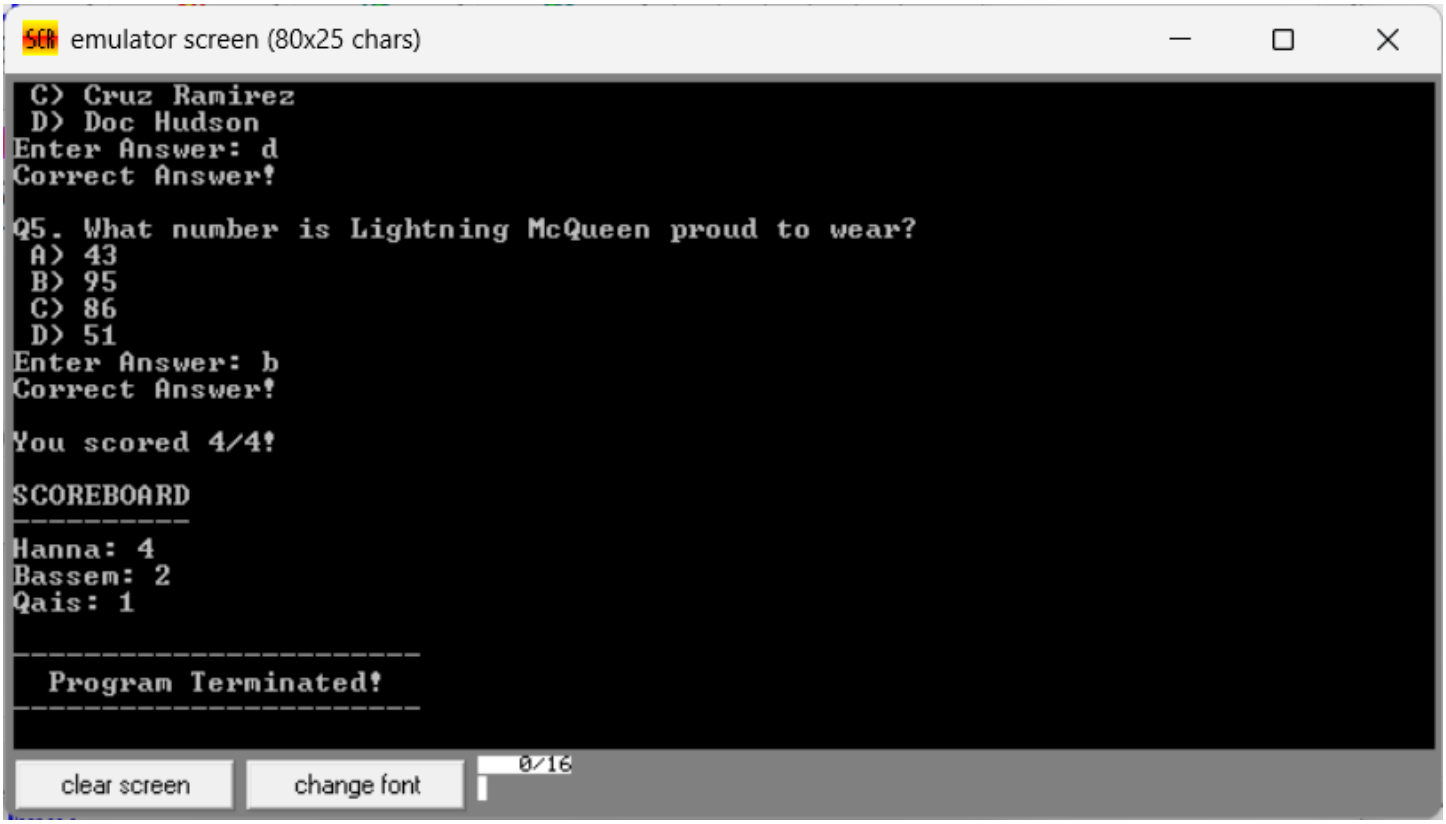

Figure 23: printScoreboard procedure output

```
; Scoreboard
    qaisScore db 48
    bassemScore db 48
    hannaScore db 48

    scoreTitle db 'SCOREBOARD$'
    scoreboardLine db '----------$'
    firstName dw ?
    firstScore db ?
    secondName dw ?
    secondScore db ?
    thirdName dw ?
    thirdScore db ?
```

Figure 24: Variables used by printScoreboard procedure in the data segment

```
; --- [Scoreboard] ---
    printScoreboard proc
        mov al, [qaisScore]
        cmp al, [bassemScore]
        jae qais_ge_bassem
        jmp bassem_gt_qais

        qais_ge_bassem:
        cmp al, [hannaScore]
        jae qais_first
        jmp hanna_first

        bassem_gt_qais:
        mov al, [bassemScore]
        cmp al, [hannaScore]
        jae bassem_first
        jmp hanna_first

        qais_first:
        lea ax, user1
        mov [firstName], ax
        mov al, [qaisScore]
        mov [firstScore], al

        mov al, [bassemScore]
        cmp al, [hannaScore]
        jae bassem_second_qais_first
        jmp hanna_second_qais_first

        bassem_second_qais_first:
        lea ax, user2
        mov [secondName], ax
        mov al, [bassemScore]
        mov [secondScore], al

        lea ax, user3
        mov [thirdName], ax
        mov al, [hannaScore]
        mov [thirdScore], al

        jmp print_sorted

        hanna_second_qais_first:
        lea ax, user3
        mov [secondName], ax
```

```
mov al, [hannaScore]
mov [secondScore], al

lea ax, user2
mov [thirdName], ax
mov al, [bassemScore]
mov [thirdScore], al

jmp print_sorted

bassem_first:
lea ax, user2
mov [firstName], ax
mov al, [bassemScore]
mov [firstScore], al

mov al, [qaisScore]
cmp al, [hannaScore]
jae qais_second_bassem_first
jmp hanna_second_bassem_first

qais_second_bassem_first:
lea ax, user1
mov [secondName], ax
mov al, [qaisScore]
mov [secondScore], al

lea ax, user3
mov [thirdName], ax
mov al, [hannaScore]
mov [thirdScore], al

jmp print_sorted

hanna_second_bassem_first:
lea ax, user3
mov [secondName], ax
mov al, [hannaScore]
mov [secondScore], al

lea ax, user1
mov [thirdName], ax
mov al, [qaisScore]
mov [thirdScore], al
```

```
    jmp print_sorted

    hanna_first:
    lea ax, user3
    mov [firstName], ax
    mov al, [hannaScore]
    mov [firstScore], al

    mov al, [qaisScore]
    cmp al, [bassemScore]
    jae qais_second_hanna_first
    jmp bassem_second_hanna_first

    qais_second_hanna_first:
    lea ax, user1
    mov [secondName], ax
    mov al, [qaisScore]
    mov [secondScore], al

    lea ax, user2
    mov [thirdName], ax
    mov al, [bassemScore]
    mov [thirdScore], al

    jmp print_sorted

    bassem_second_hanna_first:
    lea ax, user2
    mov [secondName], ax
    mov al, [bassemScore]
    mov [secondScore], al

    lea ax, user1
    mov [thirdName], ax
    mov al, [qaisScore]
    mov [thirdScore], al

    jmp print_sorted

print_sorted:
    call println

    lea dx, scoreTitle
    mov ah, 0x09
    int 21h
```

```
call println

lea dx, scoreboardLine
mov ah, 0x09
int 21h

call println

; Print first place
mov dx, [firstName]
mov ah, 0x09
int 21h

mov dl, ':'
mov ah, 0x02
int 21h

mov dl, ' '
mov ah, 0x02
int 21h

mov dl, [firstScore]
mov ah, 0x02
int 21h

call println

; Print second place
mov dx, [secondName]
mov ah, 0x09
int 21h

mov dl, ':'
mov ah, 0x02
int 21h

mov dl, ' '
mov ah, 0x02
int 21h

mov dl, [secondScore]
mov ah, 0x02
int 21h
```

```
    call println

    ; Print third place
    mov dx, [thirdName]
    mov ah, 0x09
    int 21h

    mov dl, ':'
    mov ah, 0x02
    int 21h

    mov dl, ' '
    mov ah, 0x02
    int 21h

    mov dl, [thirdScore]
    mov ah, 0x02
    int 21h

    call println

    ret
printScoreboard endp
```

Figure 25: printScoreboard Procedure

Procedure #10: terminateProgram

The **terminateProgram** procedure allows graceful exit for the program. It displays exit messages (terminate1, terminate2, terminate3) and calls int 21h/4C00h to return control to the operating system. The structured exit provides a clean program termination with no residual memory problems.

Figure 26: Output of terminateProgram procedure

```
terminate1 db '----------------------$'
terminate2 db '  Program Terminated!  $'
terminate3 db '----------------------$'
```

Figure 27: Variables used by the terminateProgram procedure in the data segment

```
; --- [Program Terimnation Logo + Exit] ---
    terminateProgram proc
        call println

        lea dx, terminate1
        mov ah, 0x09
        int 21h

        call println

        lea dx, terminate2
        mov ah, 0x09
        int 21h

        call println

        lea dx, terminate3
        mov ah, 0x09
        int 21h

        call println

        mov ax, 4c00h ; exit to operating system.
        int 21h

        ret
    terminateProgram endp
```

Figure 28: terminateProgram Procedure

Conclusion

This project implemented an MCQ Quiz Game in 8086 assembly with customizable functionality, with successful key features: a safe 3-attempts login mechanism based on family-based credentials, dynamic shuffling of 4 Cars-themed questions, case-insensitive answer checking, and real-time score update with feedback. Modular code structure was optimized for efficiency, with greater emphasis placed on hardware interaction and resource handling, while demonstrating real-world applications of low-level programming in designing interactive systems.

References

https://www.tutorialspoint.com/microprocessor/microprocessor_8086_instruction_sets.htm

https://yassinebridi.github.io/asm-docs/8086_bios_and_dos_interrupts.html#int21h

https://yassinebridi.github.io/asm-docs/8086_bios_and_dos_interrupts.html